

EXHIBIT 5

Weber, Nathan D.

From: Michael Ben-Shimon [michael@enitiatives.biz]
Sent: Sunday, August 10, 2003 5:20 AM
To: 'Ronny Sayag'
Cc: 'Gadi Erlich'
Subject: SAR-005
Attachments: SAR-005-r02.doc

Hi Ronny,

Attached please find a first draft of your invention for your review. I suggest to have another meeting to discuss your comments and some questions I still have. Please let me know a suitable time for you (I preferred this week). Please notice that this draft does not include our proposed claims, I'll add them once we conclude the detailed description of the invention.

Regards,
Michael Ben-Shimon
eNitiatives - New Business Architects Ltd.

Phone: +972-9-8890502
Mobile: +972-53-598686

This message may contain confidential information intended for the use of addressee only.

Invention Name	A Virtualization Switch and Method for Performing Virtualization in the Data-Path
Patent Disclosure Number	SAR-005
Lawyer Reference Number	
Inventor(s)	Ronny Sayag
Assignee	SanRad, Ltd.
Date	

Abstract

A virtualization switch and method for executing SCSI commands and performing virtualization in a single pass are provided. The virtualization switch optimizes the data received from the network to fit the capacity of the target volumes, thus providing higher throughput and low latency. The virtualization is preformed within the data path between the initiators and the targets and without the assistance of any devices, such as a management stations or agents installed in the initiators.

References Cited

Patents

6,209,023	Mar 2001	Dimitroff, et al.
6,519,678	Feb 2003	Basham, et al.

Published Applications

20020112113	Aug 2002	Karpoff, et al.
20030093541	May 2003	Lolayekar, et al.
20030093567	May 2003	Lolayekar, et al.
20030131182	Jul 2003	Kumar, et al.
20030140193	Jul 2003	Acharya, et al.

Other References

Technical Field

The present invention generally relates to storage area networks (SANs), and more particularly for implementation of storage virtualization in SANs.

Background of the Invention

The rapid growth in data intensive applications continues to fuel the demand for raw data storage capacity. As companies rely more and more on e-commerce, online transaction processing, and databases, the amount of information that needs to be managed and stored can be massive. As a result, the ongoing need to add more storage, service more users and back-up more data has become a challenging task.

To meet this growing demand, the concept of the storage area network (SAN) was introduced and is quickly gaining popularity. A SAN is defined as a network of which its primary purpose is the transfer of data between computer systems and storage devices. The SAN forms essentially an independent network that does not tend to have the same bandwidth limitations as its direct-connect small computer storage interface (SCSI) and network attached storage (NAS) counterparts. Specifically, in a SAN environment, storage devices and servers are generally interconnected via various switches and appliances. The connections to the switches and appliances are usually through Fiber Channel (FC). This structure generally allows for any server on the SAN to communicate with any storage device and vice versa. It also provides alternative paths from a server to a storage device. In other words, if a particular server is slow or completely unavailable, another server of the SAN can provide access to the storage device. A SAN also makes it possible to mirror data, i.e., making multiple copies available, and thus increasing reliability in the availability of data. When more storage is needed, additional storage devices can be added to the SAN without the need to be connected to a specific server.

In the related art virtualization storage architecture has been introduced to increase the utilizations of SANs, extend the scalability of storage devices, and increase the availability of data. The virtualization creates new virtual address spaces that are subsets or supersets of the address spaces of the physical storage devices. Storage virtualization architectures have two primary paths: a data path and a control path. The data path is a set of network components, devices and links that are used to transport data between servers and storage targets. The control path is a set of network links that allows network devices targets and storage transfers to be managed. A key component in virtualization storage architecture is a virtualization operator.

Reference is now made to Fig. 1, where a SAN 100 including a virtualization operator is shown. System 100 includes a virtualization operator 110, a plurality of servers 120, a FC connection 130, and a plurality of storage devices 140. Servers 120 are connected to virtualization operator 110 through network 150. The connections formed between the servers 120 and virtualization operator 110 can utilize any protocol including, but not limited to, Gigabit Ethernet carrying packets in accordance with the iSCSI protocol, TCP/IP protocol, Infiniband protocol, and others. Storage devices 140 are connected to virtualization operator 110 through FC connection 130. Storage devices 140 may include, but are not limited to, tape drives, optical drives, disks, and redundant array of inexpensive disks (RAID). Additionally, in some configurations storage devices are connected to virtualization operator 110 through FC switches. A storage device 140 is

addressable using a logical unit number (LUN). LUNs are used to identify a virtual storage that is presented by a storage subsystem or network device.

Generally, virtualization operator 110 performs the algebraic translations between a virtual address and a real storage address space. Placing virtualization operator 110 in the data path between servers 120 and storage devices 140 allows performing of in-path virtualization. That is, storage I/O transmissions between servers 120 and storage devices 140 are intercepted by virtualization operator 110 and re-transmitted to their destination. In-path virtualization, generates multiple secondary I/O requests for each incoming request. In other words, there are multiple data paths for each incoming I/O request. When I/O requests are received from servers 120, they are terminated inside virtualization operator 110. Virtualization operator 110 then creates new secondary I/O requests and transmits them to the storage targets. Finally, the results of all secondary I/O requests must be verified before acknowledging the original I/O request. The process of terminating, reinitiating and verifying I/Os adds significant latency to the storage and retrieval processes.

In the related art there are some attempts to reduce the latency and improve the performance of virtualization system architectures. For examples US patent application number 10/051,164 entitled “Serverless Storage Services” and US patent application number 10/051,415 entitled “Protocol Translation in a Storage System” disclose a storage switch (e.g., virtualization operator 110) capable of executing virtualization functions, such as mirroring, snapshot, and data replication. The storage switch is based on ingress and egress line-cards connected with a switch fabric. Each line-card includes a processing unit that carries out the virtualization functions. The storage switch performs its tasks in wire-speed. For that purpose, each line-card classifies packets into data and control packets, performs virtualization functions and protocol translation functions. The virtualization is performed without buffering data in the storage switch. That is, upon receiving the data it is immediately forwarded to a target storage device using a proprietary header attached to the data. In such implementations, the storage switch handles the entire incoming command (e.g., SCSI command) including performing the actual data transfer. This implementation is aimed at SAN architectures and is not optimized to perform virtualization functions. For example, in order to write a data block to a striped virtual volume, the SCSI command, in this case, starts in the first physical volume and ends in the second. The storage switch determines the first physical volume to which the particular data belongs and forwards the data to this volume. After completing the data transfer to the first physical volume, the data to second physical volume is retrieved from the initiator and written to the second volume. This process adds significant latency to the virtualization process.

Therefore, in the view of the limitation in the prior art it would be advantageous to provide a virtualization operator that efficiently performs virtualization services within the data path.

Brief Description of the Drawings

Figure 1 – is a SAN including a virtualization operator (prior art)

Figure 2 – is a non-limiting block diagram of a virtualization switch in accordance with an embodiment of this invention

Figure 3 – is a non-limiting functional diagram of a virtualization switch in accordance with an embodiment of this invention

Figure 4 – is an example of virtual volumes hierarchy

Figure 5 – is a non-limiting flowchart describing the method for executing a virtual read SCSI command in accordance with an embodiment of this invention

Figure 6 – is a non-limiting flowchart describing the method for executing a virtual write SCSI command in accordance with an embodiment of this invention

Figure 7 – is an exemplary diagram of a check point list

Detailed Description of the Invention

The present invention provides a virtualization switch and a method for executing SCSI commands and performing virtualization in a single pass. The virtualization switch optimizes the data received from the network to fit the capacity of the target volumes, thus providing higher throughput and low latency. The virtualization is performed within the data path between the initiators (e.g., servers 120) and the targets (e.g., storage devices 140) and without the assistance of any other devices, including, but not limited to, management stations, or agents installed in the initiators. The virtualization services that are handled by the present invention include, but are not limited to, mirroring, remote mirroring over a slow link, snapshot, data replication, striping, data concatenation, periodic local and remote and backup, restore, and the others.

Virtualization essentially means mapping of a virtual volume address space, configured by a user, to an address space on one or more physical storage target devices. A virtual volume can be anywhere on one or more physical storage devices including, but not limited to, disk, tape, and RAID, connected to a virtualization switch. Each virtual volume consists of one or more logical units (LUs), each identified by a logical unit numbers (LUN). LUNs are frequently used in the iSCSI and Fiber Channel (FC) protocols. Each LU, and hence each virtual volume, is generally comprised of one or more contiguous slices of storage space on a physical device. Thus, a virtual volume may occupy a whole storage device, a part of a single storage device, or parts of multiple storage devices. The physical storage devices, the LUs and their exact locations, are invisible to the user. In a client-server model, the target corresponds to the server, while an initiator corresponds to the client. Namely, the initiator creates and sends SCSI commands to the target as specified by a LUN.

Reference is now made to Fig. 2 where an exemplary block of a virtualization switch 200, in accordance with an embodiment of this invention, is shown. Typically, virtualization switch 200 is operated within a storage area network (i.e., SAN) and connected in the data path between the initiators and the targets. Virtualization switch 200 includes a plurality of gigabit Ethernet ports 220 and a plurality of FC ports or SCSI ports 240. Virtualization switch 200 further includes at least a processor 230, a memory 250, and a flash memory 270 connected to processor 230 by bus 280. In one embodiment, virtualization switch 200 may include a cache memory to cache data transferred through virtualization switch 200. Flash memory 270 saves the configuration of virtualization switch 200. A gigabit Ethernet port 220 is capable of carrying packets in accordance with transmission protocols including, but not limited to, iSCSI protocol, TCP/IP protocol, Infiniband protocol, and others. The network traffic, generally to be processed by virtualization switch 200, is iSCSI packets which are transported using TCP/IP packets. FC and SCSI ports 240 carry frames in accordance with the SCSI and FC protocol. Therefore, virtualization switch 200 is capable of converting incoming iSCSI packets to outgoing FC or SCSI frames, in addition to performing in the data path storage virtualization. These tasks are executed by processor 230 and will be described in greater detail below.

A typical SCSI command results in a command phase, data phase, and a response phase. In the data phase, information travels either from the initiator to the target (e.g., a WRITE command), or from the target to the initiator (e.g., a READ command). In the response phase, the target returns the final status of the operation, including any errors. A response signals the end of a typical SCSI command. The command phase includes the LUN, an initiator tag, expected data to be transferred, and command descriptor block (CDB) that embodies the SCSI command. The data phase includes a header and the actual data to be transferred. The header generally includes the LUN, the initiator tag, a data sequence number, and the number of bytes that were not transferred out of those expected to be transferred. The response phase includes a status field, used to report the SCSI status of the command, a response field that contains an iSCSI service response code, that further identifies that the command is completed or that there has been an error or failure, the number of bytes that were not transferred out of those expected to be transferred, and the number of bytes that were not transferred to the initiator out of those expected to be transferred.

Reference is now made to Fig. 3 where a functional diagram of virtualization switch 200 in accordance with an embodiment of this invention, is shown. Virtualization switch includes the following components: a network interface (NI) 310, an iSCSI module 320, a target manager (TM) 330, a data transfer arbiter (DTA) 340, a volume manger (VM) 350, and a device manger (DM) 360.

NI 310 interfaces between TCP/IP network (e.g., network 150) and virtualization switch 200 through Ethernet ports 220. NI 310 includes Ethernet MAC drivers and a TCP/IP stack (not shown) which accelerates the TCP/IP packets processing. The iSCSI module 320 includes an iSCSI stack implementing the iSCSI protocol.

TM 330 implements the SCSI level logic, i.e., TM 330 parses the incoming SCSI commands to determine the type of the commands, the LUN, and the number of bytes to be transferred. An incoming SCSI command refers to a virtual volume. TM 330 schedules the execution of SCSI commands according to a predefined scheduling algorithm and generates data transport requests to DTA 340. For that purpose, each of the incoming SCSI commands is kept in a queue related to a specific LU. In addition, TM 330 creates and holds all targets (i.e., physical storage devices) exposed to virtualization switch 200. A target may be connected to virtualization switch 200 through more than one port 240, where each connection defines a different target path. When a SCSI command should be sent to one of the targets, the command is sent via one of the target paths assigned to this target. In one embodiment, TM 330 may perform load balancing between the target paths. TM 330, further interacts with VM 350 for the purpose of executing SCSI commands that are not requires any data transfer ,e.g., check the status of a virtual volume.

DTA 340 performs the actual data transfer between targets and the initiators. DTA 340 together with VM 350 translates the SCSI logical commands to physical commands, i.e., performing the virtualization. Specifically, each logical command is converted to a list of

physical commands structured in a proprietary data structure. A detailed example, for the virtualization process is provided below.

VM 350 executes all the virtualization operations. That is, each request to a virtual volume is directed to VM 350 that in return provides a list of physical commands that are sent to the physical storage devices. Each physical command includes the physical address in a single physical storage device indicating where to write the data or from where to obtain the data. To execute the virtualization operations VM 350 maintains a mapping schema that defines relations between the virtual volumes, the LUs, and the physical storage devices. The virtual volume may include, but are not limited to, a concatenation volume, a stripped volume, a mirrored volume, and a snapshot volume. As an example, the virtual volume 410-A shown in Fig. 4A is a mirror set of two other virtual volumes 410-B and 410-C. Virtual volume 410-B is a concatenation of two LUs 420-1 and 420-2. Virtual volume 410-C is stripped over three different LUs 420-3, 420-4, and 420-5. A LU is defined as a plurality of continuous data blocks having the same block size. The blocks are addressable by the user and, by definition, reside in address spaces different from the address spaces of physical storage devices. For instance, the virtual address spaces of virtual volumes 410-A , 410-B, and 410-C are 0000-1000, 1000-2000, and 2000-3000, respectively. The address spaces of LUs 420-1, 420-2, 420-3, 420-4, and 420-5 are 1000-1600, 1600-2000, 2000-2200, 2200-2700, and 2700-3000 respectively. The physical address spaces of the storage occupied by LUs 420-1, 420-2, 420-3, 420-4, and 420-5 is denoted by the physical address of the data blocks. As mentioned above, VM 350 together with DTA 340 generate a data structure which includes a list of physical commands. This data structure includes the address spaces of the virtual volumes, and the LUs, the connections between the LUs and virtual volumes, the physical addresses of the actual data, and pointers to the physical storage devices. Fig. 4B shows a non-limiting exemplary data structure formed in accordance with the volumes array shown in Fig. 4A.

DM 360 handles the operation of SCSI devices connected to virtualization switch 220 using parallel SCSI or FC. DM 360 includes at least two host bus adapters (HBAs) the first HBA interacts with a simple disk controller and the second HBA interacts with a RAID controller. The HBA drivers implement a simple generic SCSI interface, enabling the sending and receiving of SCSI commands with error recovery services. The interface also provides basic discovery services and exposes a list of SCSI target paths to the upper layer. DM 360 maintains a list of targets (i.e., physical storage devices) and a list of targets' paths associated with each target. In addition, for each target a LU discovery process is performed. For each discovered LU a LUN is assigned.

It should be appreciated by a person skilled in the art that the components of virtualization switch 200 described herein may be hardware components, firmware components, software components, or combination thereof.

Reference is now made to Fig. 5 where a non-limiting flowchart 500 describing the method for executing a Read SCSI command in accordance with an embodiment of this invention, is shown.

At step S510, TCP/IP packets are received and processed by means network interface 310. At step S520, a new SCSI command is received at iSCSI module 320 which initiates an iSCSI session with an initiator (e.g., one of servers 120). If the new incoming SCSI command was transmitted by an initiator which is not registered in virtualization switch 200, then the initiator identification (ID) is passed to TM 330 for registration. TM 330 may deny the registration of the new initiator, if the new initiator is unauthorized to access virtualization switch 200. At step S530 the new SCSI command is sent to TM 330, which parses the SCSI command from an upper-level message. At step S540, a check is performed to determine if the incoming SCSI command is valid. If the SCSI command is invalid, then at step S545 a response command, including the iSCSI service code, is generated and sent to the initiator. Otherwise, execution continues with step S550 where the incoming SCSI command is added to a queue that holds tasks waiting to be executed on the LU specified in the command. TM 330 schedules the tasks execution using a scheduling algorithm, such as recently used, round robin, weighted round robin, random, least loaded LU, or any other applicable algorithm. At step S555, when the command is scheduled to be executed, TM 330 transfers the SCSI command to DTA 340. At step S560, DTA 340 together with VM 350 translate the logical SCSI command to a list of physical commands. The conversion is performed in one pass and result in a data structure including the list of physical commands. To allow for load balancing each physical command further includes the number of bytes expected to be read from each target. At step S565, the physical commands are sent to iSCSI module 320. At step S570, iSCSI module 320 provides DTA 340 with the current number of bytes that can be transferred to the initiator. This is preformed in order to optimize the data transferred through the network. At step S580, DAT 340 using DM 360 retrieves data equals to the number of bytes that can be transferred to the initiator. At step S585, DTA 380 transfers the retrieved data to iSCSI module 320 which subsequently sends the data to the initiator. At step S590, DTA 340 performs a check to determine if the entire data requested to be read was transferred to the initiator. If more data is to be read, then execution continues with step S570, otherwise execution continues with step S595. At step S595, DTA 340 informs iSCSI module 320 that the entire requested data has been transferred. In addition, DTA 340 informs TM 330 that the command execution has ended. As a result, TM 330 removes the command from the queue and iSCSI model 320 sends a response command to the initiator. The response command signals the end of the SCSI command.

Reference is now made to Fig. 6 where a non-limiting flowchart 600 describing the method for execution of a Write SCSI command in accordance with an embodiment of this invention, is shown.

At step S610, TCP/IP packets are received data from the network (e.g., network 150). The received packets are processed by means network interface 310. At step S620, a new logical SCSI command is received at iSCSI module 320, which subsequently initiates an iSCSI session with the command's initiator (e.g., one of servers 120). If the new incoming SCSI command was transmitted by an initiator which is not registered in virtualization switch 200, then the initiator identification (ID) is passed to TM 330 for registration. TM 330 may deny the registration of the new initiator, if the new initiator is

unauthorized to access virtualization switch 200. At step S630, the new SCSI command is sent to TM 330, which parses the SCSI command from an upper-level message. At step S640, a check is performed to determine if the incoming SCSI command is valid. If the SCSI command is invalid, then at step S645 a response command including the iSCSI service code is generated and sent to the initiator. Otherwise, execution continues with step S650 where the incoming SCSI command is added to a queue that holds tasks waiting to be executed on the LU designated in the incoming command. TM 330 schedules the tasks execution using a scheduling algorithm, such as recently used, round robin, weighted round robin, random, least loaded LU, or any other applicable algorithm. At step S655, when the command is scheduled to be executed, TM 330 transfers the SCSI command to DTA 340. At step S660, DTA 340 together with VM 350 translate the logical SCSI command to a list of physical commands. The conversion is performed in one pass as described in greater detail above. DTA 340 further generates a check-point list which describes how the data should be delivered from the initiator to DTA 340. The check-point list is a list of data chunks that DTA 340 expected to receive from the initiator. Fig. 7 shows an exemplary checkpoint list 700 for writing a data block having the size of 750 bytes to virtual address space 2000-2750. This data block has to be written to three different LUs (and hence to three different storage devices) 410-3, 410-4, and 410-5. Therefore, check-point list 700 should include at least three data chunks. Check-point list 700 includes five data chunks 710 through 750, where the size of data chunks 710, 720, 730, 740, and 750 are 100 bytes, 100 bytes, 200 bytes, 300 bytes, and 50 bytes respectively. Data chunks 710 and 720 are targeted to LU 410-3, data chunks 730 and 750 are targeted to LU 410-4, and data chunk 740 is targeted to LU 410-5. A size of a data chunk is defined by a minimum and maximum number of bytes and may be changed from one chunk to another. Each data chunk may target to a different physical storage. The check-point list is used to avoid situations where initiator sends a very small chunk of data (e.g., one byte) to DTA 340. Such situations can easily overload DTA 340 and thus reduce performance. The iSCSI module 320 fills the data chunks in check-point list and each data chunk at a time. At step S670, iSCSI module 320 aggregates the data bytes received from the initiator until the number of received bytes can fill the i^{th} data chunk in the check-point list. Each time, a single data chunk is filled with data beginning from the first data chunk and ending in the last data chunk in the check-point list. At step S680, the i^{th} data chunk is sent to DTA 340, which subsequently sends the data chunk to the physical target. At step S685, DM 360 acknowledges that the data chunk was written to the physical target. At step S690, DTA 340 performs a check to determine if all the data chunks were written to the target physical storages. If more data chunks need to be written, then execution continues with step S670, otherwise execution continues with step S695. At step S695, DTA 340 informs iSCSI module 320 that the entire requested data was transferred. In addition, DTA 340 informs TM 330 that the command execution ended. As a result, TM 330 removes the command from the queue and iSCSI model 320 sends a response command to the initiator. The response command signals the end of the SCSI command.

Virtualization switch 200 buffers the data before writing and reading from or to the physical storage devices. In one embodiment the memory buffers used are scatter gather

lists (SGLs) that are composed of multiple data segments linked together using a linked list. A SGL represents a logical contiguous buffer.

In one embodiment the present invention there is provided a software mechanism that determines which errors should be reported by the virtual volume. Specifically, a virtual volume consists of more than one physical volume. Hence, an error generated by only one of the physical volumes may not affect the functionality of the virtual volume and therefore this error should not be reported to the user. To determine whether or not to report a virtual volume error, the virtualization switch 200 aggregates the errors produced by the physical volumes for each of the virtual volumes. Based on the number of errors, the error types, and decision criteria, virtualization switch 200 determines whether or not to report a virtual volume error. The decision criteria may be defined by the user or may be set dynamically according to the currently available resources.

What We Seek to Protect

1. A virtualization switch
2. A method for performing virtualization within the data path
3. A computer executable code for performing virtualization within the data path

Proposed Claims

<TBD>

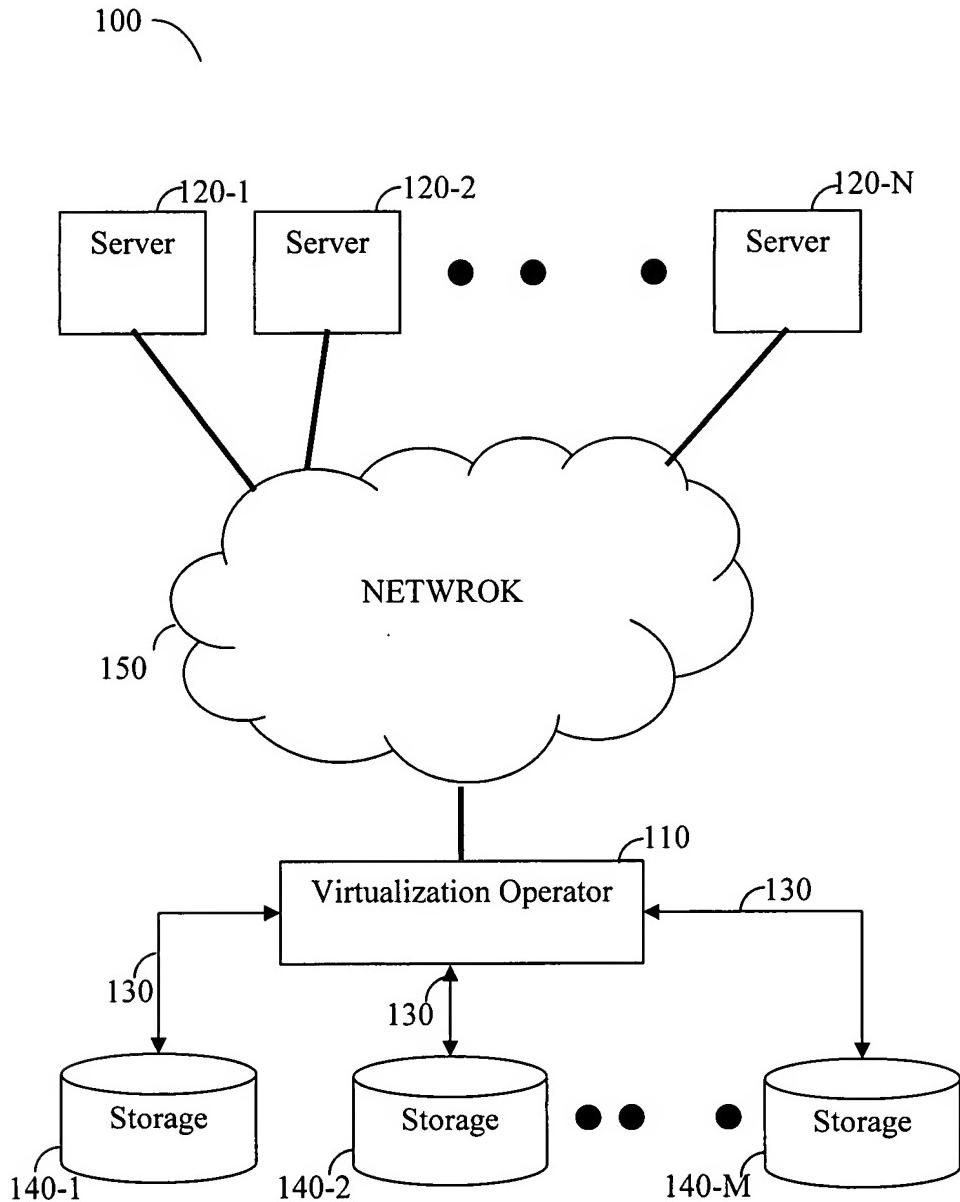
Figures

FIGURE 1

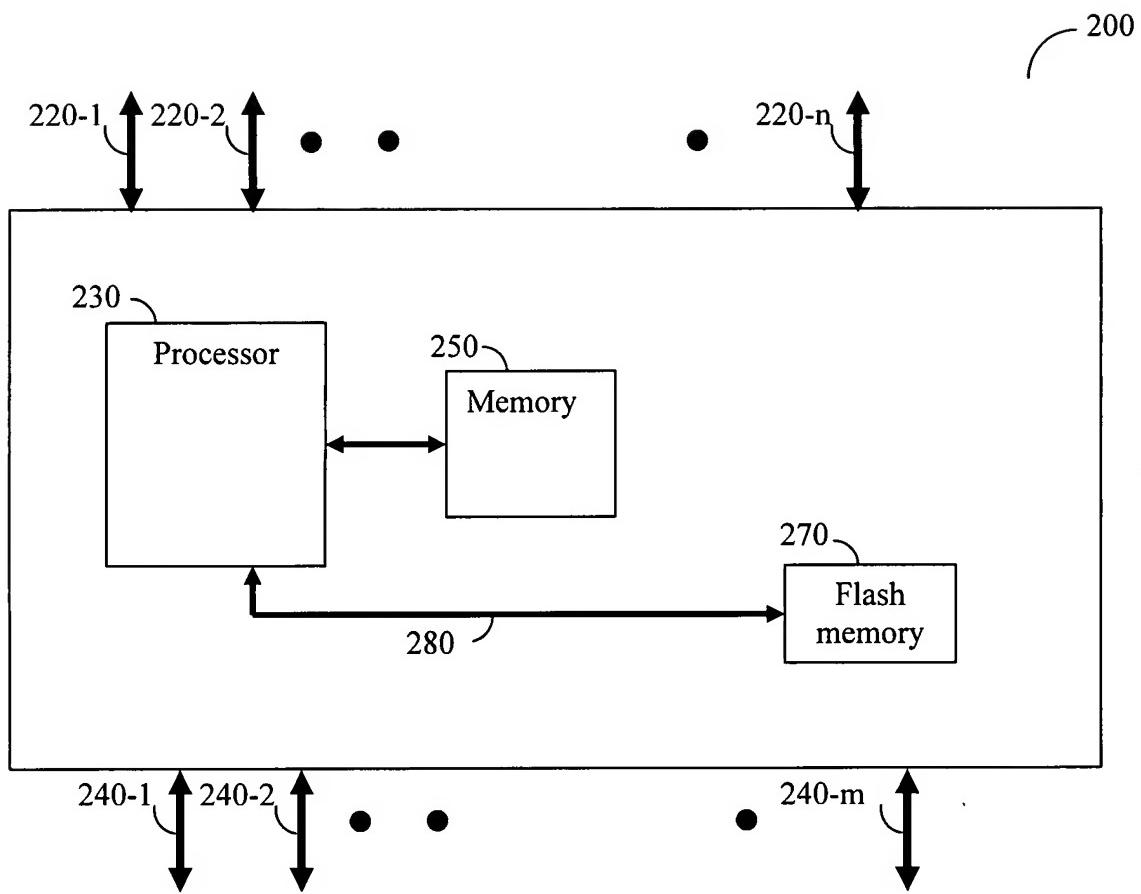


FIGURE 2

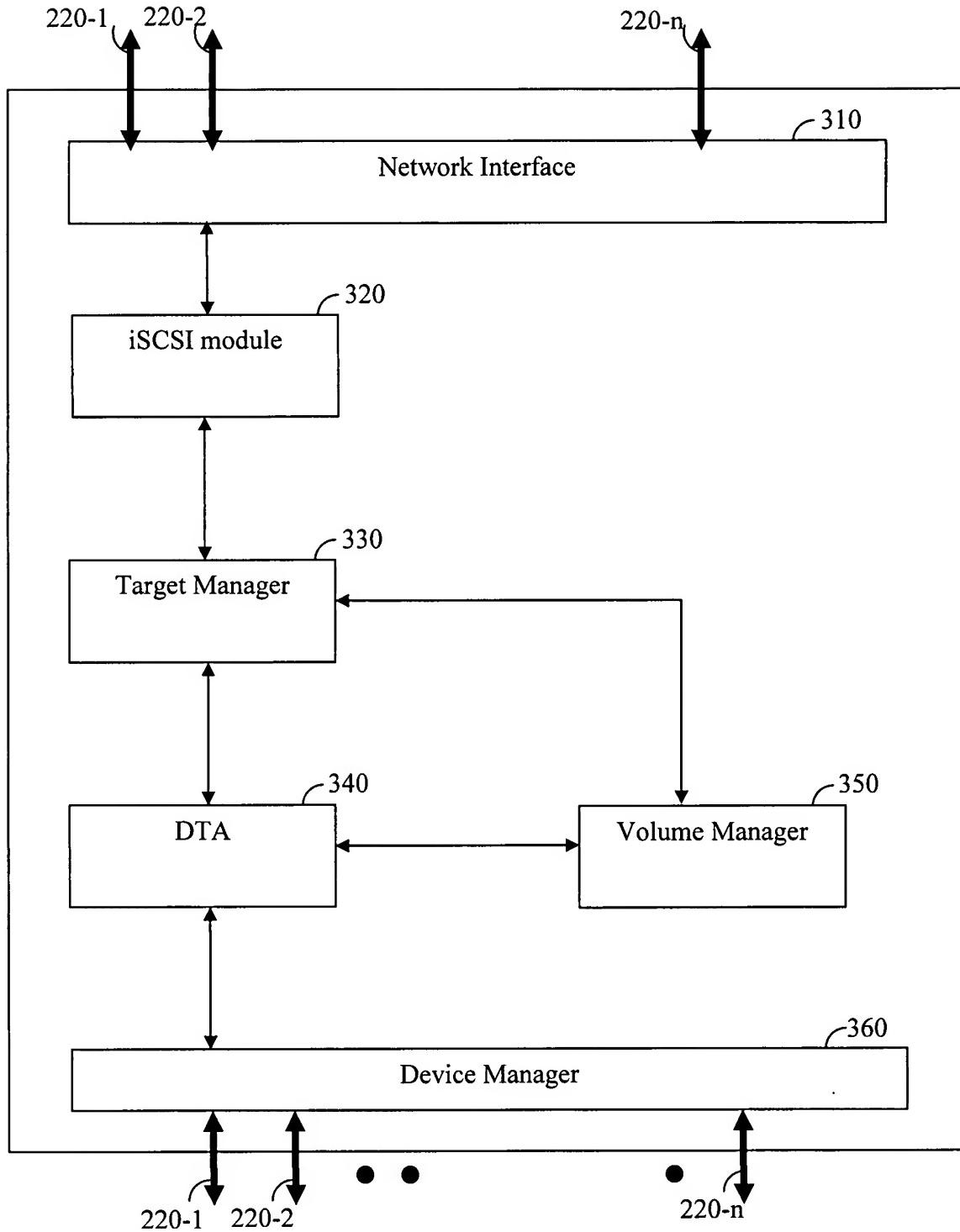


FIGURE 3

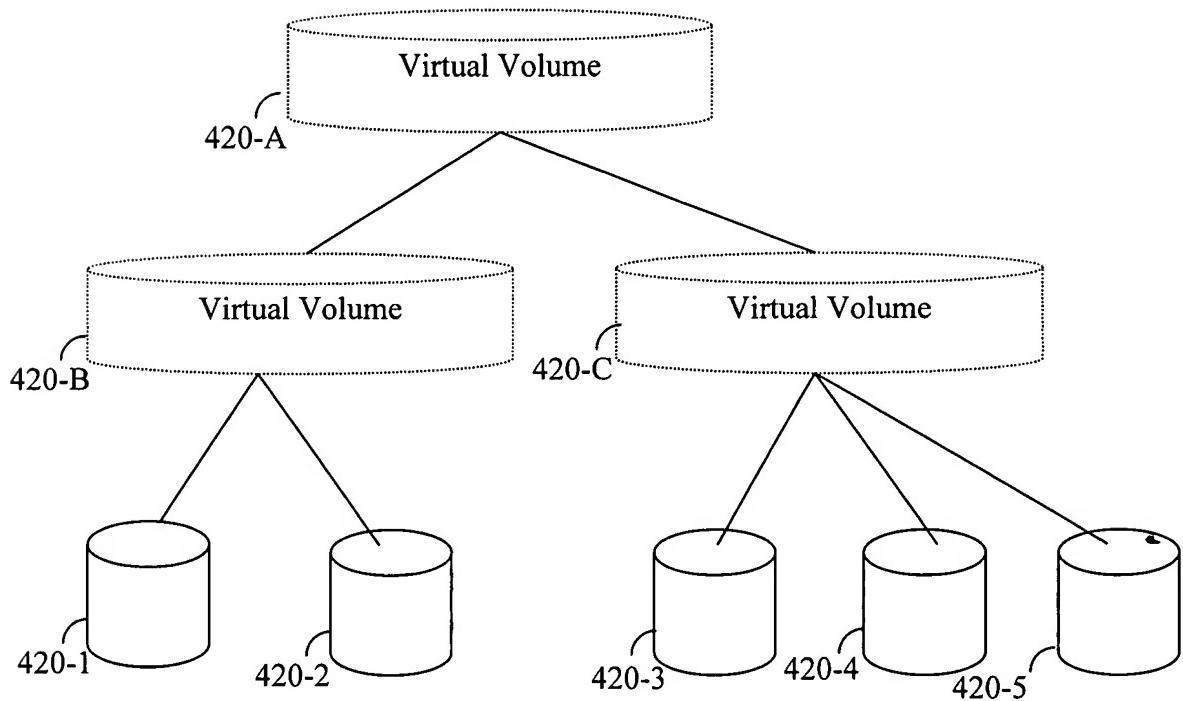


FIGURE 4A

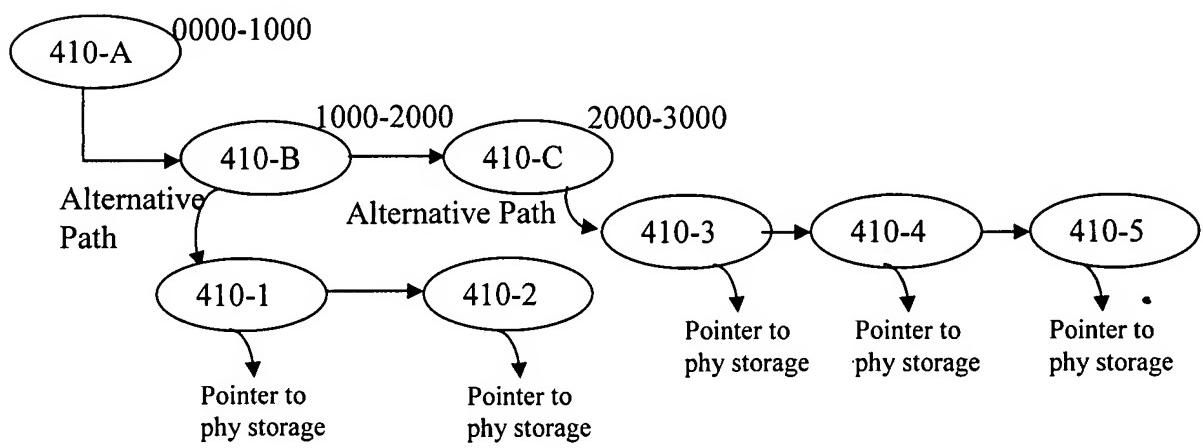


FIGURE 4B

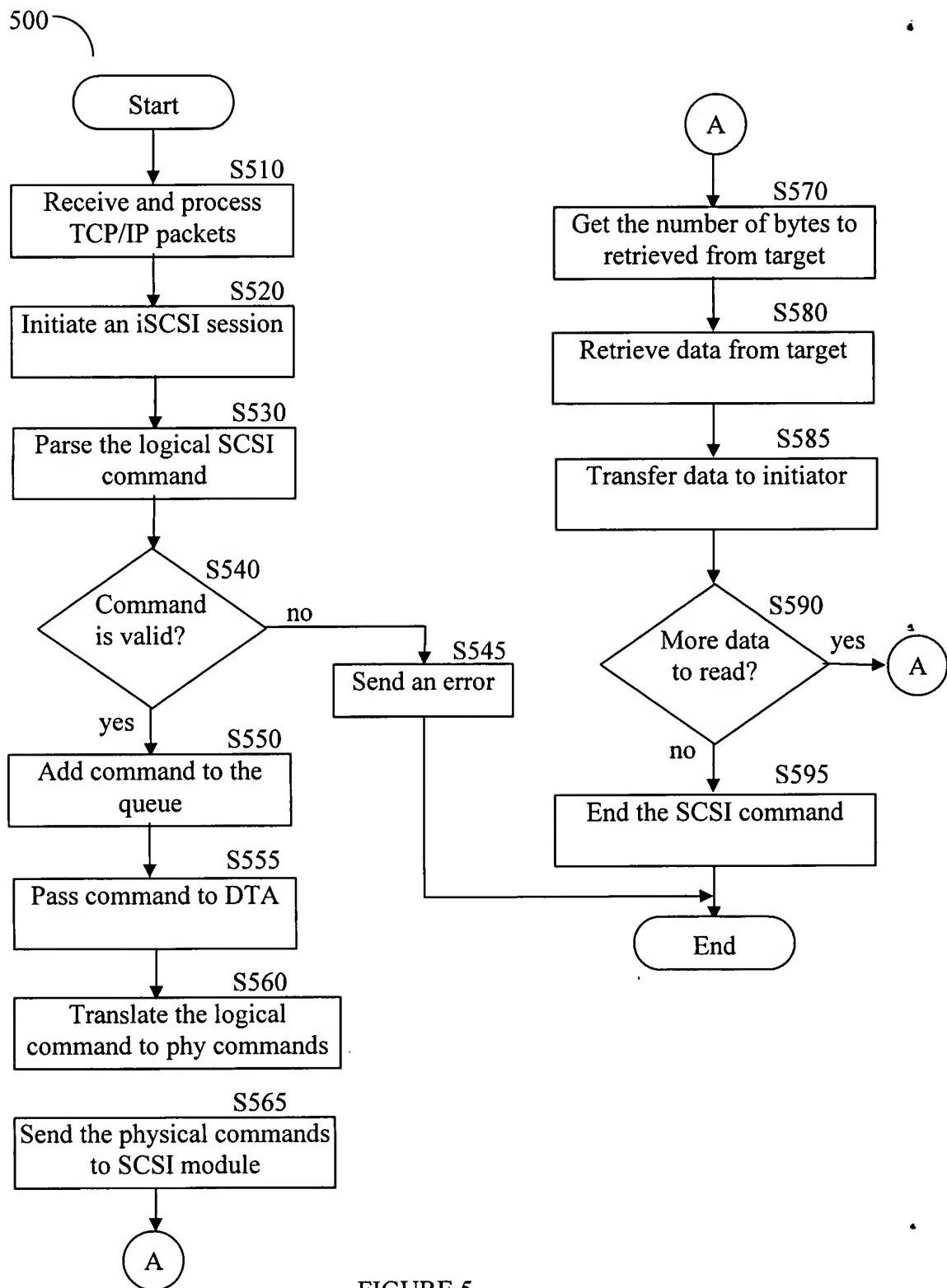


FIGURE 5

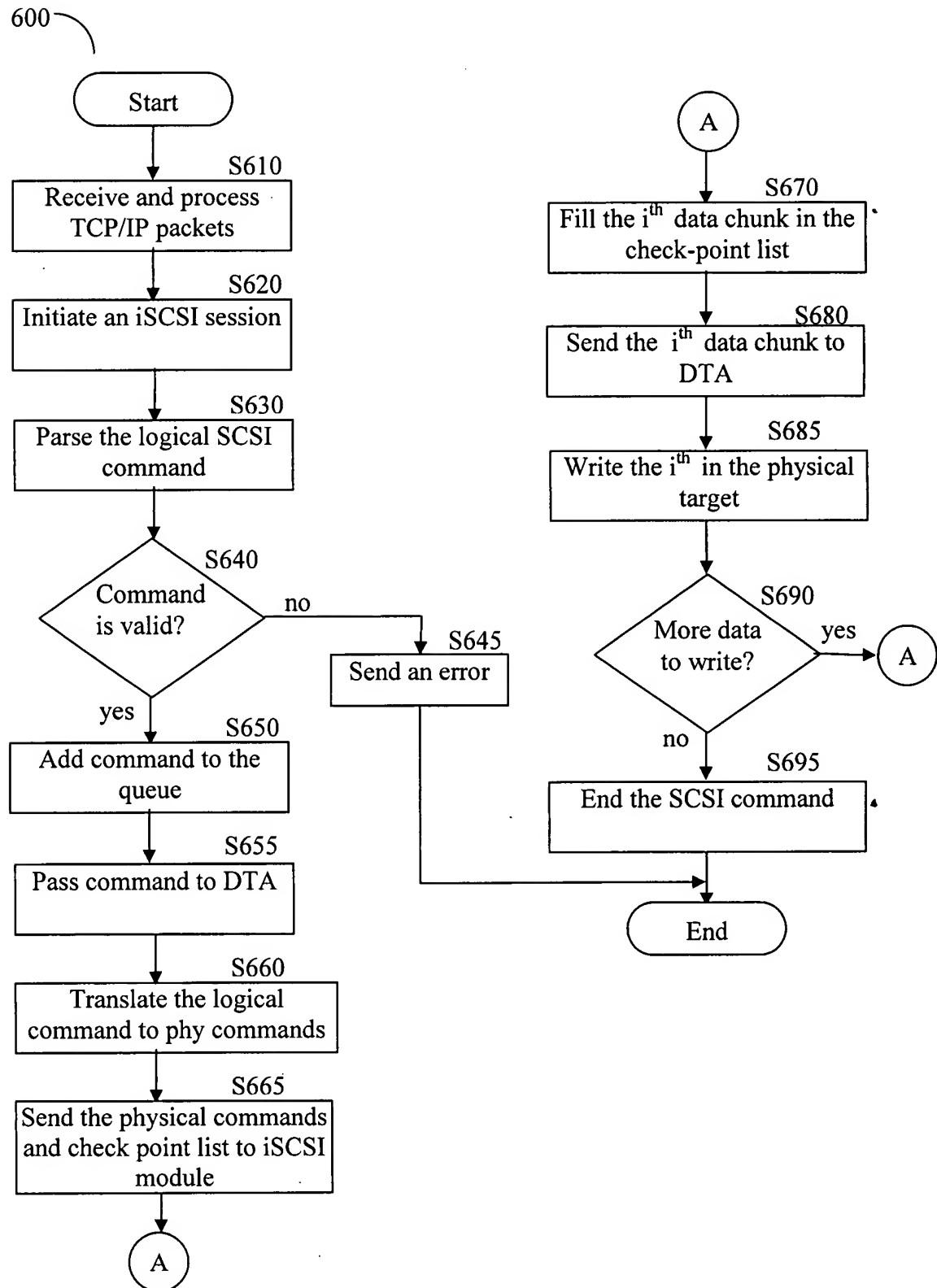


FIGURE 6

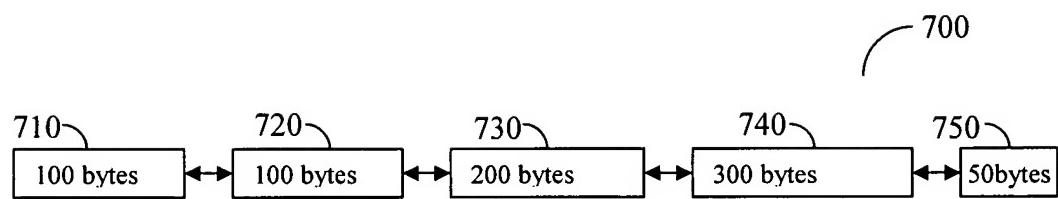


FIGURE 7